Concurrent Programming for Scalable Web Architectures

Diploma Thesis
VS-D01-2012

Institute of Distributed Systems
Faculty of Engineering and Computer Science
Ulm University

Benjamin Erb

April 20, 2012

2.3 Concurrency

Concurrency is a property of a system representing the fact that multiple activities are executed at the same time. According to Van Roy [Roy04], a program having "several independent activities, each of which executes at its own pace". In addition, the activities may perform some kind of interaction among them. The concurrent execution of activities can take place in different environments, such as single-core processors, multi-core processors, multi-processors or even on multiple machines as part of a distributed system. Yet, they all share the same underlying challenges: providing mechanisms to control the different flows of execution via coordination and synchronization, while ensuring consistency.

Apart from recent hardware trends towards multi-core and multiprocessor systems, the use of concurrency in applications is generally motivated by performance gains. Cantrill et al. [Can08] describe three fundamental ways how the concurrent execution of an application can improve its performance:

Reduce latency A unit of work is executed in shorter time by subdivision into parts that can be executed concurrently.

Hide latency Multiple long-running tasks are executed together by the underlying system. This is particularly effective when the tasks are blocked because of external resources they must wait upon, such as disk or network I/O operations.

Increase throughput By executing multiple tasks concurrently, the general system throughput can be increased. It is important to notice that this also speeds up independent sequential tasks that have not been specifically designed for concurrency yet.

The presence of concurrency is an intrinsic property for any kind of distributed system. Processes running on different machines form a common system that executes code on multiple machines at the same time.

Conceptually, all web applications can be used by various users at the same time. Thus, a web application is also inherently concurrent. This is not limited to the web server that must handle multiple client connections in parallel. Also the application that executes the associated business logic of a request and the backend data storage are confronted with concurrency.

2.3.1 Concurrency and Parallelism

In the literature, there are varying definitions of the terms concurrency and parallelism, and their relation. Sometimes both terms are even used synonymously. We will now introduce a distinction of both terms and of their implications on programming.

Concurrency vs. Parallelism

Concurrency is a conceptual property of a program, while parallelism is a runtime state. Concurrency of a program depends on the programming language and the way it is coded, while parallelism depends on the actual runtime environment. Given two tasks to be executed concurrently, there are several possible execution orders. They may be performed sequentially (in any order), alternately, or even simultaneously. Only executing two different tasks simultaneously yields true parallelism. In terms of scheduling, parallelism can only be achieved if the hardware architecture supports parallel execution, like multi-core or multi-processor systems do. A single-core machine will also be able to execute multiple threads concurrently, however it can never provide true parallelism.

Concurrent Programming vs. Parallel Programming

Differentiating concurrent and parallel programming is more tedious, as both are targeting different goals on different conceptual levels. Concurrent programming tackles concurrent and interleaving tasks and the resulting complexity due to a nondeterministic control flow. Reducing and hiding latency is equally important to improving throughput. Instead, parallel programming favors a deterministic control flow and mainly reaches for optimized throughput. The internals of a web server are the typical outcome of concurrent programming, while the parallel abstractions such as Google's `MapReduce` [Dea08] or Java's `fork/join` [Lea00] provide a good example of what parallel programming is about. Parallel programming is also essential for several specialized tasks. For instance, a graphics processing unit is designed for massive floating-point computational power and usually runs a certain numerical computation in parallel on all of its units at the same time. High-performance computing is another important area of parallel programming. It takes advantage of computer clusters and distributes sub-tasks to cluster nodes, thus speeding up complex computations.

Computer Architectures Supporting Concurrency

The previous differentiation can also be backed by a closer look on the architectures where physical concurrency is actually possible. We will therefore refer to Flynn's taxonomy [Fly72], which is shown in table 2.3. This classification derives four different types of computer architectures, based on the instruction concurrency and the available data streams. The Single Instruction,

	Single Instruction	**Multiple Instruction**
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

Table 2.3: Flynn's taxonomy classifying different computer architectures.

Single Data stream (SISD) class is represented by traditional single processor machines. We do not consider them further due to their lack of physical concurrency. Multiple Instruction, Single Data stream (MISD) is a rather exotic architecture class, sometimes used for fault-tolerant computing where the same instructions are executed redundantly. It is also not relevant for our considerations. Real parallelism can only be exploited on architectures that support multiple data streams—Single Instruction, Multiple Data streams (SIMD) and Multiple Instruction, Multiple Data streams (MIMD). SIMD represents the aforementioned architecture class targeting dedicated parallel executions of computations such as graphics processing unit and vector processors. SIMD exploits data-level parallelism which is not suitable for handling web requests. Accordingly, this type of architecture is not relevant for our consideration. We will focus on the last remaining class, MIMD. It represents architectures that rely on a shared or distributed memory and thus includes architectures possessing multiple cores, multiple CPUs or even multiple machines. In the following, we will primarily focus on concurrent programming (parallel execution of subtasks is partially relevent in chapter 5) and only on the MIMD class of architectures.

2.3.2 Models for Programming Concurrency

Van Roy [Roy04] introduces four main approaches for programming concurrency that we will examine briefly (see figure 2.1). The more important models will be studied later as potential solutions for programming concurrency inside web architectures, especially in chapter 5.

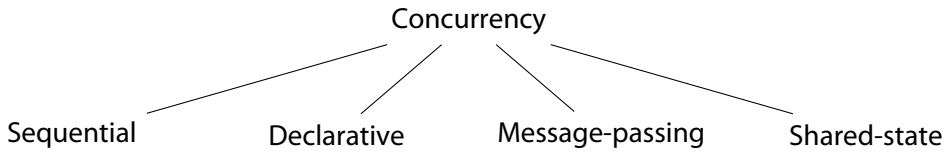

Figure 2.1: Models for Programming Concurrency (Van Roy [Roy04])

Sequential Programming

In this deterministic programming model, no concurrency is used at all. In its strongest form, there is a total order of all operations of the program. Weaker forms still keep the deterministic behaviour. However, they either make no guarantees on the exact execution order to the programmer *a priori*. Or they provide mechanisms for explicit preemption of the task currently active, as co-routines do, for instance.

Declarative Concurrency

Declarative programming is a programming model that favors implicit control flow of computations. Control flow is not described directly, it is rather a result of computational logic of the

program. The declarative concurrency model extends the declarative programming model by allowing multiple flows of executions. It adds implicit concurrency that is based on a data-driven or a demand-driven approach. While this introduces some form of nondeterminism at runtime, the nondeterminism is generally not observable from the outside.

Message-passing Concurrency

This model is a programming style that allows concurrent activities to communicate via messages. Generally, this is the only allowed form of interaction between activities which are otherwise completely isolated. Message passing can be either synchronous or asynchronous resulting in different mechanisms and patterns for synchronization and coordination.

Shared-state Concurrency

Shared-state concurrency is an extended programming model where multiple activities are allowed to access contended resources and states. Sharing the exact same resources and states among different activities requires dedicated mechanisms for synchronization of access and coordination between activities. The general nondeterminism and missing invariants of this model would otherwise directly cause problems regarding consistency and state validity.

2.3.3 Synchronization and Coordination as Concurrency Control

Regardless of the actual programming model, there must be an implicit or explicit control over concurrency (at least within the runtime environment). It is both hazardous and unsafe when multiple flows of executions simultaneously operate in the same address space without any kind of agreement on ordered access. Two or more activities might access the same data and thus induce data corruption as well as inconsistent or invalid application state. Furthermore, multiple activities that work jointly on a problem need an agreement on their common progress. Both issues represent fundamental challenges of concurrency and concurrent programming.

Synchronization and *coordination* are two mechanisms attempting to tackle this. Synchronization, or more precisely *competition synchronization* as labeled by Sebesta [Seb05], is a mechanism that controls access on shared resources between multiple activities. This is especially important when multiple activities require access to resources that cannot be accessed simultaneously. A proper synchronization mechanism enforces exclusiveness and ordered access on the resource by different activities. Coordination, sometimes also named *cooperation synchronization* (Sebesta [Seb05]), aims at the orchestration of collaborating activities.

Synchronization and coordination are sometimes conflated in practice. Both mechanisms can be either implicit or explicit (Van Roy [Roy04]). *Implicit synchronization* hides the synchronization as part of the operational semantics of the programming language. Thus, it is not part of the visible program code. On the contrary, *explicit synchronization* requires the programmer to add explicit synchronization operations to the code.

2.3.4 Tasks, Processes and Threads

So far, our considerations of concurrency were based on computer architectures and programming models. We will now show how they interrelate by introducing the actual activity entities used and provided by operating systems and how they are mapped to the hardware. Note that we will use the term *task* as a general abstraction for a unit of execution from now on.

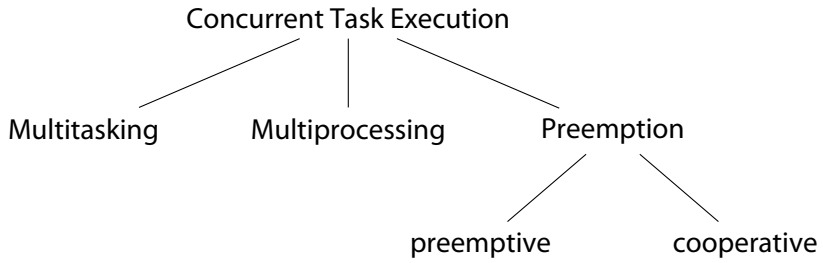

Figure 2.2: Orthogonal concepts for the concurrent execution of multiple tasks.

The ability to execute multiple tasks concurrently has been a crucial requirement for operating systems. It is addressed by *multitasking*. This mechanism manages an interleaved and alternating execution of tasks. In case of multiple CPU cores or CPUs, multitasking can be complemented by *multiprocessing*, which allocates different tasks on available cores/CPUs. The key concept for both mechanisms is *scheduling*, which organizes the assignment of processing time to tasks using *scheduling strategies*. Appropriate strategies can have different aims, such as fair scheduling between tasks, fixed latencies for task executions or maximum utilization. Another distinction of scheduling strategies is their preemption model. In the *preemptive* model, the scheduler assigns processing time to a task and eventually revokes it. The task has no control over the preemption. In a *cooperative* model, the task itself has the responsibility of yielding after some time to allow another task to run. Scheduling is an important duty of any operating system. However, it is also noteworthy that applications themselves can provide some kind of scheduling of their own internal tasks, as we will see in chapter 4 and 5.

Operating systems generally provide different types of tasks—processes and threads. Essentially, they represent different task granularities. A *process* is a heavyweight task unit and owns system resources such as memory and file descriptors that are allocated from the operating system. *Threads* are leightweight task units that belong to a certain process. All threads allocated within the same process share memory, file descriptors and other related resources. Creating threads is a relatively cheap operation compared to the creation of new processes.

Most concurrent applications make heavy use of multiple threads. However, this does not imply the availability of threads as explicit entities of the programming language itself. Instead, the execution environment might map other concurrency constructs of a programming language to actual threads at runtime. Similarly, the shared-state property of multithreading might be idiomatically hidden by the language.

2.3.5 Concurrency, Programming Languages and Distributed Systems

Next, we consider the strong relationship between concurrent programming, programming languages and distributed systems when building large architectures. Programming distributed systems introduces a set of additional challenges compared to regular programming. The "Fallacies of Distributed Computing" [RGO06] provide a good overview on some of the important pitfalls that must be addressed. For instance, not anticipating errors in a network or ignoring the different semantics of local and remote operation are common misconceptions that are described.

From a software engineering perspective, the major challenges are fault tolerance, integration of distribution aspects and reliability. As we have already seen before, distributed systems are inherently concurrent and parallel, thus concurrency control is also essential.

Programming languages to be used for distributed systems must either incorporate appropriate language idioms and features to meet these requirements. Otherwise, frameworks are necessary to provide additional features on top of the core language.

Ghosh et al. [Gho11] have considered the impact of programming languages on distributed systems. They pointed out that mainstream languages like Java and C++ are still the most popular choice of developing distributed systems. They are combined with middleware frameworks most of the time, providing additional features. However, the strengths of general purpose languages do not cover the main requirements of distributed systems to a great extent. The experiences with RPC-based systems (see Kendall et al. [Ken94]) and their object-based descendents (see Vinoski [Vin08]) have raised some questions to this approach. Middleware systems providing distributability compensate for features missing at the core of a language. Thus, the systems actually meet the necessary requirements, but they are often also cumbersome to use and introduce superfluous complexity.

Recently, there has been an increasing interest in various alternative programming languages embracing high-level concurrency and distributed computing. Being less general, these languages focus on important concepts and idioms for distributed systems, such as component abstractions, fault tolerance and distribution mechanisms. It is interesting for our considerations that most of these languages oriented towards distributed computing also incorporate alternative concurrency approaches. We will have a brief look on some of these languages as part of chapter 5.

2.4 Scalability

Scalability is a non-functional property of a system that describes the ability to appropriately handle increasing (and decreasing) workloads. According to Coulouris et al. [Dol05], "a system is described as scalable, if it will remain effective when there is a significant increase in the number of resources and the number of users". Sometimes, scalability is a requirement that necessitates the usage of a distributed system in the first place. Also, scalability is not to be confused with raw speed or performance. Scalability competes with and complements other non-functional requirements such as availability, reliability and performance.

Bibliography

[Aba10] Abadi, Daniel: Problems with CAP, and Yahoo's little known NoSQL system, Blog Post: http://dbmsmusings.blogspot.com/2010/04/problems-with-cap-and-yahoos-little.html (2010)

[Abb09] Abbott, Martin L. and Fisher, Michael T.: *The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise*, Addison-Wesley Professional (2009)

[Abb11] Abbott, Martin L. and Fisher, Michael T.: *Scalability Rules: 50 Principles for Scaling Web Sites*, Addison-Wesley Professional (2011)

[Ady02] Adya, Atul; Howell, Jon; Theimer, Marvin; Bolosky, William J. and Douceur, John R.: Cooperative Task Management Without Manual Stack Management, in: *Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference*, USENIX Association, Berkeley, CA, USA, pp. 289–302

[Aga07] Agarwal, Aditya; Slee, Mark and Kwiatkowski, Marc: Thrift: Scalable Cross-Language Services Implementation, Tech. Rep., Facebook (2007)

[Agh90] Agha, Gul: Concurrent object-oriented programming. *Commun. ACM* (1990), vol. 33:pp. 125–141

[All08] Allspaw, John: *The Art of Capacity Planning: Scaling Web Resources*, O'Reilly Media (2008)

[All10] Allspaw, John and Robbins, Jesse: *Web Operations: Keeping the Data On Time*, O'Reilly Media (2010)

[Alv11] Alvaro, Peter; Conway, Neil; Hellerstein, Joe and Marczak, William R.: Consistency Analysis in Bloom: a CALM and Collected Approach, in: *CIDR*, pp. 249–260

[AMQ11] AMQP Working Group: AMQP Specification v1.0, Tech. Rep., Organization for the Advancement of Structured Information Standards (2011)

[And10] Anderson, J. Chris; Lehnardt, Jan and Slater, Noah: *CouchDB: The Definitive Guide: Time to Relax (Animal Guide)*, O'Reilly Media (2010)

[Arm07] Armstrong, Joe: *Programming Erlang: Software for a Concurrent World*, Pragmatic Bookshelf (2007)

[Bak77] Baker, Henry G., Jr. and Hewitt, Carl: The Incremental Garbage Collection of Processes, Tech. Rep., Cambridge, MA, USA (1977)

[Bel09] Belshe, Mike: SPDY: An experimental protocol for a faster web, Tech. Rep., Google Inc. (2009)

[Ber81] Bernstein, Philip A. and Goodman, Nathan: Concurrency Control in Distributed Database Systems. *ACM Comput. Surv.* (1981), vol. 13(2):pp. 185–221

[BL01] Berners-Lee, Tim; Hendler, James and Lassila, Ora: The Semantic Web. *Scientific American* (2001), vol. 284(5):pp. 34–43

[BL05] Berners-Lee, T.; Fielding, R. and Masinter, L.: Uniform Resource Identifier (URI): Generic Syntax, RFC 3986 (Standard) (2005)

[Blo08] Bloch, Joshua: *Effective Java (2nd Edition)*, Addison-Wesley (2008)

[Bra08] Bray, Tim; Paoli, Jean; Maler, Eve; Yergeau, François and Sperberg-McQueen, C. M.: Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C recommendation, W3C (2008), http://www.w3.org/TR/2008/REC-xml-20081126/

[Bre00] Brewer, Eric A.: Towards robust distributed systems (abstract), in: *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, PODC '00, ACM, New York, NY, USA, pp. 7–

[Bro87] Brooks, Frederick P., Jr.: No Silver Bullet Essence and Accidents of Software Engineering. *Computer* (1987), vol. 20(4):pp. 10–19

[Can08] Cantrill, Bryan and Bonwick, Jeff: Real-World Concurrency. *Queue* (2008), vol. 6:pp. 16–25

[Cas08] Cascaval, Calin; Blundell, Colin; Michael, Maged; Cain, Harold W.; Wu, Peng; Chiras, Stefanie and Chatterjee, Siddhartha: Software Transactional Memory: Why Is It Only a Research Toy? *Queue* (2008), vol. 6:pp. 46–58

[Cha06] Chang, Fay; Dean, Jeffrey; Ghemawat, Sanjay; Hsieh, Wilson C.; Wallach, Deborah A.; Burrows, Mike; Chandra, Tushar; Fikes, Andrew and Gruber, Robert E.: Bigtable: A distributed storage system for structured data, in: *In Proceedings Of The 7Th Conference On Usenix Symposium On Operating Systems Design And Implementation - Volume 7*, pp. 205–218

[Cle04] Clement, Luc; Hately, Andrew; von Riegen, Claus and Rogers, Tony: UDDI Spec Technical Committee Draft 3.0.2, Oasis committee draft, OASIS (2004)

[Cod70] Codd, E. F.: A relational model of data for large shared data banks. *Commun. ACM* (1970), vol. 13(6):pp. 377–387

[Con63] Conway, Melvin E.: Design of a separable transition-diagram compiler. *Commun. ACM* (1963), vol. 6(7):pp. 396–408

[Cre09] Creeger, Mache: Cloud Computing: An Overview. *Queue* (2009), vol. 7:pp. 2:3–2:4

[Cro06] Crockford, D.: The application/json Media Type for JavaScript Object Notation (JSON), RFC 4627 (Informational) (2006)

[Cro08] Crockford, Douglas: *JavaScript: The Good Parts*, Yahoo Press (2008)

[Dah09] Dahan, Udi: Clarified CQRS, Tech. Rep., udidahan.com (2009)

[Dea08] Dean, Jeffrey and Ghemawat, Sanjay: MapReduce: simplified data processing on large clusters. *Commun. ACM* (2008), vol. 51:pp. 107–113

[DeC07] DeCandia, Giuseppe; Hastorun, Deniz; Jampani, Madan; Kakulapati, Gunavardhan; Lakshman, Avinash; Pilchin, Alex; Sivasubramanian, Swaminathan; Vosshall, Peter and Vogels, Werner: Dynamo: amazon's highly available key-value store, in: *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, ACM, New York, NY, USA, pp. 205–220

[Dij65] Dijkstra, Edsger Wybe: Cooperating Sequential Processes, Technical Report EWD-123, Tech. Rep., University of Texas at Austin (1965)

[Dol05] Dollimore, Jean; Kindberg, Tim and Coulouris, George: *Distributed Systems: Concepts and Design (4th Edition)*, Addison Wesley (2005)

[Dra11] Dragojević, Aleksandar; Felber, Pascal; Gramoli, Vincent and Guerraoui, Rachid: Why STM can be more than a research toy. *Commun. ACM* (2011), vol. 54:pp. 70–77

[ECM99] ECMA International: Standard ECMA-262, Tech. Rep., ECMA International (1999)

[Eis99] Eisenberg, Andrew and Melton, Jim: SQL: 1999, formerly known as SQL3. *SIGMOD Rec.* (1999), vol. 28(1):pp. 131–138

[Esw76] Eswaran, K. P.; Gray, J. N.; Lorie, R. A. and Traiger, I. L.: The notions of consistency and predicate locks in a database system. *Commun. ACM* (1976), vol. 19(11):pp. 624–633

[Fet11] Fette, I. and Melnikov, A.: The WebSocket Protocol, RFC 6455 (Informational) (2011)

[Fie99] Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P. and Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1, RFC 2616 (Draft Standard) (1999), updated by RFCs 2817, 5785, 6266

[Fie00] Fielding, Roy Thomas: *Architectural styles and the design of network-based software architectures*, Ph.D. thesis, University of California, Irvine (2000), aAI9980887

[Fly72] Flynn, Michael J.: Some computer organizations and their effectiveness. *IEEE Trans. Comput.* (1972), vol. 21(9):pp. 948–960

[For12] Foresti, A.; Singhal, S.; Mazahir, O.; Nielsen, H.; Raymor, B.; Rao, R. and Montenegro, G.: HTTP Speed+Mobility, Tech. Rep., Microsoft (2012)

[Fow02] Fowler, Martin: *Patterns of Enterprise Application Architecture*, Addison-Wesley Professional (2002)

[Fow05] Fowler, Martin: Event Sourcing, Tech. Rep., ThoughtWorks (2005)

[Fri76] Friedman, Daniel and Wise, David: The Impact of Applicative Programming on Multiprocessing, in: *International Conference on Parallel Processing*

[Fri99] Frigo, Matteo; Leiserson, Charles E.; Prokop, Harald and Ramachandran, Sridhar: Cache-Oblivious Algorithms, in: *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, IEEE Computer Society, Washington, DC, USA, pp. 285–

[fS86] for Standardization, International Organization: *ISO 8879:1986: Information processing — Text and office systems — Standard Generalized Markup Language (SGML)*, International Organization for Standardization (1986)

[Gho10] Ghosh, Debasish: *DSLs in Action*, Manning Publications (2010)

[Gho11] Ghosh, Debasish; Sheehy, Justin; Thorup, Kresten and Vinoski, Steve: Programming language impact on the development of distributed systems. *Journal of Internet Services and Applications* (2011), vol. Issue 2 / 2011:pp. 1–8, 10.1007/s13174-011-0042-y

[Gif79] Gifford, David K.: Weighted voting for replicated data, in: *Proceedings of the seventh ACM symposium on Operating systems principles*, SOSP '79, ACM, New York, NY, USA, pp. 150–162

[Gil02] Gilbert, Seth and Lynch, Nancy: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News* (2002), vol. 33:pp. 51–59

[Gm92] Garcia-molina, Hector and Salem, Kenneth: Main memory database systems: An overview. *IEEE Transactions on Knowledge and Data Engineering* (1992), vol. 4:pp. 509–516

[Goe06] Goetz, Brian; Peierls, Tim; Bloch, Joshua; Bowbeer, Joseph; Holmes, David and Lea, Doug: *Java Concurrency in Practice*, Addison-Wesley Professional (2006)

[Goe10] Goetz, Brian: JSR 335: Lambda Expressions for the Java(TM) Programming Language, Tech. Rep., Oracle Inc. (2010)

[Gos12] Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad and Buckley, Alex: The Java Language Specification, Java SE 7 Edition, Tech. Rep., Oracle Inc. (2012)

[Gro07] Grossman, Dan: The transactional memory / garbage collection analogy, in: *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*, OOPSLA '07, ACM, New York, NY, USA, pp. 695–706

[Gud07] Gudgin, Martin; Hadley, Marc; Mendelsohn, Noah; Lafon, Yves; Moreau, Jean-Jacques; Karmarkar, Anish and Nielsen, Henrik Frystyk: SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C recommendation, W3C (2007), http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

[Gue07] Guerraoui, Rachid: A Smooth Concurrency Revolution with Free Objects. *IEEE Internet Computing* (2007), vol. 11:pp. 82–85

[Gus05] Gustafsson, Andreas: Threads without the Pain. *Queue* (2005), vol. 3:pp. 34–41

[Hae83] Haerder, Theo and Reuter, Andreas: Principles of transaction-oriented database recovery. *ACM Comput. Surv.* (1983), vol. 15(4):pp. 287–317

[Hal06] Haller, Philipp and Odersky, Martin: Event-Based Programming without Inversion of Control, in: David E. Lightfoot and Clemens A. Szyperski (Editors) *Modular Programming Languages*, Lecture Notes in Computer Science, pp. 4–22

[Hal08] Haller, Philipp and Odersky, Martin: Scala actors: Unifying thread-based and event-based programming. *Theoretical Computer Science* (2008)

[Har08] Harris, Tim; Marlow, Simon; Jones, Simon Peyton and Herlihy, Maurice: Composable memory transactions. *Commun. ACM* (2008), vol. 51:pp. 91–100

[Hel07] Hellerstein, Joseph M.; Stonebraker, Michael and Hamilton, James: *Architecture of a Database System*, Now Publishers Inc., Hanover, MA, USA (2007)

[Hel09] Helland, Pat and Campbell, David: Building on Quicksand, in: *CIDR*

[Hel10] Hellerstein, Joseph M.: The declarative imperative: experiences and conjectures in distributed logic. *SIGMOD Rec.* (2010), vol. 39:pp. 5–19

[Hel12] Helland, Pat: Idempotence Is Not a Medical Condition. *Queue* (2012), vol. 10(4):pp. 30:30–30:46

[Her93] Herlihy, Maurice and Moss, J. Eliot B.: Transactional memory: architectural support for lock-free data structures, in: *Proceedings of the 20th annual international symposium on computer architecture*, ISCA '93, ACM, New York, NY, USA, pp. 289–300

[Hew73] Hewitt, Carl; Bishop, Peter and Steiger, Richard: A universal modular ACTOR formalism for artificial intelligence, in: *Proceedings of the 3rd international joint conference on Artificial intelligence*, IJCAI'73, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 235–245

[Hic08] Hickey, Rich: The Clojure programming language, in: *Proceedings of the 2008 symposium on Dynamic languages*, DLS '08, ACM, New York, NY, USA, pp. 1:1–1:1

[Hic09a] Hickson, Ian: The Web Sockets API, W3C working draft, W3C (2009), http://www.w3.org/TR/2009/WD-websockets-20091222/

[Hic09b] Hickson, Ian: Web Storage, Last call WD, W3C (2009), http://www.w3.org/TR/2009/WD-webstorage-20091222/

[Hic09c] Hickson, Ian: Web Workers, Last call WD, W3C (2009), http://www.w3.org/TR/2009/WD-workers-20091222/

[Hoa74] Hoare, C. A. R.: Monitors: an operating system structuring concept. *Commun. ACM* (1974), vol. 17(10):pp. 549–557

[Hoa78] Hoare, C. A. R.: Communicating sequential processes. *Commun. ACM* (1978), vol. 21(8):pp. 666–677

[Hoh03] Hohpe, Gregor and Woolf, Bobby: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Professional (2003)

[Hoh06] Hohpe, Gregor: Programming Without a Call Stack – Event-driven Architectures, Tech. Rep., eaipatterns.com (2006)

[Hya09] Hyatt, David and Hickson, Ian: HTML 5, W3C working draft, W3C (2009), http://www.w3.org/TR/2009/WD-html5-20090825/

[Ire09] Ireland, Christopher; Bowers, David; Newton, Michael and Waugh, Kevin: A Classification of Object-Relational Impedance Mismatch, in: *Proceedings of the 2009 First International Conference on Advances in Databases, Knowledge, and Data Applications*, DBKDA '09, IEEE Computer Society, Washington, DC, USA, pp. 36–43

[Jac99] Jacobs, Ian; Raggett, David and Hors, Arnaud Le: HTML 4.01 Specification, W3C recommendation, W3C (1999), http://www.w3.org/TR/1999/REC-html401-19991224

[Kar97] Karger, David; Lehman, Eric; Leighton, Tom; Panigrahy, Rina; Levine, Matthew and Lewin, Daniel: Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web, in: *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, ACM, New York, NY, USA, pp. 654–663

[Keg06] Kegel, Dan: The C10K problem, Tech. Rep., Kegel.com (2006)

[Ken94] Kendall, Samuel C.; Waldo, Jim; Wollrath, Ann and Wyant, Geoff: A Note on Distributed Computing, Tech. Rep., Sun Microsystems Laboratories, Mountain View, CA, USA (1994)

[Kni86] Knight, Tom: An architecture for mostly functional languages, in: *Proceedings of the 1986 ACM conference on LISP and functional programming*, LFP '86, ACM, New York, NY, USA, pp. 105–112

[Lam78] Lamport, Leslie: Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* (1978), vol. 21(7):pp. 558–565

[Lam79a] Lampson, B. and Sturgis, H.: Crash recovery in a distributed storage system, Tech. Rep., Xerox Palo Alto Research Center, Palo Alto, CA (1979)

[Lam79b] Lampson, Butler W. and Redell, David D.: Experience with processes and monitors in Mesa (Summary), in: *Proceedings of the seventh ACM symposium on Operating systems principles*, SOSP '79, ACM, New York, NY, USA, pp. 43–44

[Lam98] Lamport, Leslie: The part-time parliament. *ACM Trans. Comput. Syst.* (1998), vol. 16(2):pp. 133–169

[Lar08] Larson, Jim: Erlang for Concurrent Programming. *Queue* (2008), vol. 6:pp. 18–23

[Lau79] Lauer, Hugh C. and Needham, Roger M.: On the duality of operating system structures. *SIGOPS Oper. Syst. Rev.* (1979), vol. 13:pp. 3–19

[Lea00] Lea, Doug: A Java fork/join framework, in: *Proceedings of the ACM 2000 conference on Java Grande*, JAVA '00, ACM, New York, NY, USA, pp. 36–43

[Lee06] Lee, Edward A.: The Problem with Threads. *Computer* (2006), vol. 39:pp. 33–42

[Les09] Lesani, Mohsen; Odersky, Martin and Guerraoui, Rachid: Transactors: Unifying Transactions and Actors, Tech. Rep. (2009)

[Les11] Lesani, Mohsen and Palsberg, Jens: Communicating memory transactions, in: *Proceedings of the 16th ACM symposium on Principles and practice of parallel programming*, PPoPP '11, ACM, New York, NY, USA, pp. 157–168

[Li07] Li, Peng and Zdancewic, Steve: Combining events and threads for scalable network services implementation and evaluation of monadic, application-level concurrency primitives, in: *Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, PLDI '07, ACM, New York, NY, USA, pp. 189–199

[Lis88] Liskov, B. and Shrira, L.: Promises: linguistic support for efficient asynchronous procedure calls in distributed systems. *SIGPLAN Not.* (1988), vol. 23(7):pp. 260–267

[Liu07] Liu, Canyang Kevin and Booth, David: Web Services Description Language (WSDL) Version 2.0 Part 0: Primer, W3C recommendation, W3C (2007), http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626

[Mar12] Marr, Stefan and D'Hondt, Theo: Identifying A Unifying Mechanism for the Implementation of Concurrency Abstractions on Multi-Language Virtual Machines, in: *Objects, Models, Components, Patterns, 50th International Conference, TOOLS 2012*, Springer, (to appear)

[Mas98] Masinter, L.: The "data" URL scheme, RFC 2397 (Proposed Standard) (1998)

[Mat09] Mattson, Robert L. R. and Ghosh, Somnath: HTTP-MPLEX: An enhanced hypertext transfer protocol and its performance evaluation. *J. Netw. Comput. Appl.* (2009), vol. 32(4):pp. 925–939

[McG11] McGranaghan, Mark: ClojureScript: Functional Programming for JavaScript Platforms. *IEEE Internet Computing* (2011), vol. 15(6):pp. 97–102

[Mey01] Meyer, Eric A. and Bos, Bert: CSS3 introduction, W3C working draft, W3C (2001), http://www.w3.org/TR/2001/WD-css3-roadmap-20010523/

[Moi11] Moiz, Salman Abdul; P., Sailaja; G., Venkataswamy and Pal, Supriya N.: Article: Database Replication: A Survey of Open Source and Commercial Tools. *International Journal of Computer Applications* (2011), vol. 13(6):pp. 1–8, published by Foundation of Computer Science

[Mol03] Molnar, Ingo: The Native POSIX Thread Library for Linux, Tech. Rep., Tech. Rep., RedHat, Inc (2003)

[Mos06] Moseley, Ben and Marks, Peter: Out of the Tar Pit, Tech. Rep. (2006)

[Mur10] Murray, Derek G. and Hand, Steven: Scripting the cloud with skywriting, in: *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud'10, USENIX Association, Berkeley, CA, USA, pp. 12–12

[Mur11] Murray, Derek G.; Schwarzkopf, Malte; Smowton, Christopher; Smith, Steven; Madhavapeddy, Anil and Hand, Steven: CIEL: a universal execution engine for distributed data-flow computing, in: *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, USENIX Association, Berkeley, CA, USA, pp. 9–9

[Not11] Nottingham, Mark: On HTTP Load Testing, Blog Post: http://www.mnot.net/blog/2011/05/18/http_benchmark_rules (2011)

[Not12] Nottingham, Mark: What's Next for HTTP, Blog Post: http://www.mnot.net/blog/2012/03/31/whats_next_for_http (2012)

[Oka96] Okasaki, Chris: *Purely Functional Data Structures*, Ph.D. thesis, Carnegie Mellon University (1996)

[Ous96] Ousterhout, John: Why Threads are a Bad Idea (for most purposes), in: *USENIX Winter Technical Conference*

[Pai99] Pai, Vivek S.; Druschel, Peter and Zwaenepoel, Willy: Flash: an efficient and portable web server, in: *Proceedings of the annual conference on USENIX Annual Technical Conference*, USENIX Association, Berkeley, CA, USA, pp. 15–15

[Par07] Pariag, David; Brecht, Tim; Harji, Ashif; Buhr, Peter; Shukla, Amol and Cheriton, David R.: Comparing the performance of web server architectures, in: *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, ACM, New York, NY, USA, pp. 231–243

[Pri08] Pritchett, Dan: BASE: An Acid Alternative. *Queue* (2008), vol. 6(3):pp. 48–55

[Pya97] Pyarali, Irfan; Harrison, Tim; Schmidt, Douglas C. and Jordan, Thomas D.: Proactor - An Object Behavioral Pattern for Demultiplexing and Dispatching Handlers for Asynchronous Events, Tech. Rep., Washington University (1997)

[Ran78] Randell, B.; Lee, P. and Treleaven, P. C.: Reliability Issues in Computing System Design. *ACM Comput. Surv.* (1978), vol. 10(2):pp. 123–165

[Ran10] Randles, Martin; Lamb, David and Taleb-Bendiab, A.: A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing, in: *Proceedings of the 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*, WAINA '10, IEEE Computer Society, Washington, DC, USA, pp. 551–556

[Ree78] Reed, D. P.: *Naming And Synchronization In A Decentralized Computer System*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, USA (1978)

[RGO06] Rotem-Gal-Oz, Arnon; Gosling, James and Deutsch, L. Peter: Fallacies of Distributed Computing Explained, Tech. Rep., Sun Microsystems (2006)

[Rob04] Robinson, D. and Coar, K.: The Common Gateway Interface (CGI) Version 1.1, RFC 3875 (Informational) (2004)

[Ros11] Rose, John: SR 292: Supporting Dynamically Typed Languages on the Java(TM) Platform, Tech. Rep., Oracle America, Inc. (2011)

[Roy04] Roy, Peter Van and Haridi, Seif: *Concepts, Techniques, and Models of Computer Programming*, The MIT Press (2004)

[Rum11] Rumble, Stephen M.; Ongaro, Diego; Stutsman, Ryan; Rosenblum, Mendel and Ousterhout, John K.: It's time for low latency, in: *Proceedings of the 13th USENIX conference on Hot topics in operating systems*, HotOS'13, USENIX Association, Berkeley, CA, USA, pp. 11–11

[Rys11] Rys, Michael: Scalable SQL. *Commun. ACM* (2011), vol. 54(6):pp. 48–53

[Sch95] Schmidt, Douglas C.: *Reactor: an object behavioral pattern for concurrent event demultiplexing and event handler dispatching*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1995), pp. 529–545

[Sch06] Schlossnagle, Theo: *Scalable Internet Architectures*, Sams (2006)

[Sch08] Schwartz, Baron; Zaitsev, Peter; Tkachenko, Vadim; D., Jeremy Zawodny; Lentz, Arjen and Balling, Derek J.: *High Performance MySQL: Optimization, Backups, Replication, and More*, O'Reilly Media (2008)

[Seb05] Sebesta, Robert W.: *Concepts of Programming Languages (7th Edition)*, Addison Wesley (2005)

[Sha00] Shachor, Gal and Milstein, Dan: The Apache Tomcat Connector - AJP Protocol Reference, Tech. Rep., Apache Software Foundation (2000)

[Ske82] Skeen, Dale: A Quorum-Based Commit Protocol, Tech. Rep., Cornell University, Ithaca, NY, USA (1982)

[Ske83] Skeen, D. and Stonebraker, M.: A Formal Model of Crash Recovery in a Distributed System. *IEEE Trans. Softw. Eng.* (1983), vol. 9(3):pp. 219–228

[Ste03] Stevens, W. Richard; Fenner, Bill and Rudoff, Andrew M.: *Unix Network Programming, Volume 1: The Sockets Networking API (3rd Edition)*, Addison-Wesley Professional (2003)

[Sto07] Stonebraker, Michael; Madden, Samuel; Abadi, Daniel J.; Harizopoulos, Stavros; Hachem, Nabil and Helland, Pat: The end of an architectural era: (it's time for a complete rewrite), in: *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, VLDB Endowment, pp. 1150–1160

[Sto10] Stonebraker, Michael: Errors in Database Systems, Eventual Consistency, and the CAP Theorem, Web Article: http://bit.ly/cCDWDS (2010)

[Sut05] Sutter, Herb and Larus, James: Software and the Concurrency Revolution. *Queue* (2005), vol. 3:pp. 54–62

[Tan06] Tanenbaum, Andrew S. and Steen, Maarten Van: *Distributed Systems: Principles and Paradigms (2nd Edition)*, Prentice Hall (2006)

[Tea12] Team, The Dart: Dart Programming Language Specification, Tech. Rep., Google Inc. (2012)

[Tha10] Thalinger, Christian and Rose, John: Optimizing invokedynamic, in: *Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java*, PPPJ '10, ACM, New York, NY, USA, pp. 1–9

[Tho11] Thompson, Martin; Farley, Dave; Barker, Michael; Gee, Patricia and Stewart, Andrew: Disruptor: High performance alternative to bounded queues for exchanging data between concurrent threads, Tech. Rep., LMAX (2011)

[Til10] Tilkov, Stefan and Vinoski, Steve: Node.js: Using JavaScript to Build High-Performance Network Programs. *IEEE Internet Computing* (2010), vol. 14:pp. 80–83

[Ung10] Ungar, David and Adams, Sam S.: Harnessing emergence for manycore programming: early experience integrating ensembles, adverbs, and object-based inheritance, in: *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, SPLASH '10, ACM, New York, NY, USA, pp. 19–26

[vB03a] von Behren, Rob; Condit, Jeremy and Brewer, Eric: Why events are a bad idea (for high-concurrency servers), in: *Proceedings of the 9th conference on Hot Topics in Operating Systems - Volume 9*, USENIX Association, Berkeley, CA, USA, pp. 4–4

[vB03b] von Behren, Rob; Condit, Jeremy; Zhou, Feng; Necula, George C. and Brewer, Eric: Capriccio: scalable threads for internet services, in: *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, ACM, New York, NY, USA, pp. 268–281

[Vin07] Vinoski, Steve: Concurrency with Erlang. *IEEE Internet Computing* (2007), vol. 11:pp. 90–93

[Vin08] Vinoski, Steve: Convenience Over Correctness. *IEEE Internet Computing* (2008), vol. 12:pp. 89–92

[Vin09] Vinoski, Steve: Welcome to "The Functional Web". *IEEE Internet Computing* (2009), vol. 13:pp. 104–103

[Vog08] Vogels, Werner: Eventually Consistent. *Queue* (2008), vol. 6(6):pp. 14–19

[Wel01] Welsh, Matt; Culler, David and Brewer, Eric: SEDA: an architecture for well-conditioned, scalable internet services, in: *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, ACM, New York, NY, USA, pp. 230–243

[Wel10] Welsh, Matt: A Retrospective on SEDA, Blog Post: http://matt-welsh.blogspot.com/2010/07/retrospective-on-seda.html (2010)

[Win99] Winer, Dave: XML-RPC Specification, Tech. Rep., UserLand Software, Inc. (1999)

[Zah10] Zaharia, Matei; Chowdhury, Mosharaf; Franklin, Michael J.; Shenker, Scott and Stoica, Ion: Spark: cluster computing with working sets, in: *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud'10, USENIX Association, Berkeley, CA, USA, pp. 10–10

[Zel03] Zeldovich, Nickolai; Yip, Er; Dabek, Frank; Morris, Robert T.; Mazières, David and Kaashoek, Frans: Multiprocessor support for event-driven programs, in: *In Proceedings of the 2003 USENIX Annual Technical Conference (USENIX '03*, pp. 239–252

[Zhu11] Zhu, Wenbo: Implications of Full-Duplex HTTP, Tech. Rep., Google, Inc. (2011)